Construction of Efficient Decision Trees

AUTHOR(S):

MIYAKAWA, Masahiro

CITATION:

MIYAKAWA, Masahiro. Construction of Efficient Decision Trees. 数理解析研究所講究録 1988, 666: 285-301

ISSUE DATE:

1988-07

URL:

http://hdl.handle.net/2433/100661

数理解析研究所講究録
第 666 巻 1988 年 285-301

Construction of Efficient Decision Trees

Masahiro MIYAKAWA
(宮 川 正 弘)

Electrotechnical Laboratory
1-1-4 Umezono, Tsukuba, Japan 305

Abstract

Construction algorithms of optimum and near-optimum decision trees are surveyed under two optimality criteria: d-cost (e.g. number of the nodes of a tree) and e-cost (e.g. average time of testing for a decision). Special attentions are paid for: 1) presenting new selection criteria of a variable for constructing near-optimum trees, 2) exploring their properties and 3) the comparison of the performance of the criteria. Experimental results are also mentioned, giving some conclusive remarks about the performance.

For converting a decision table by the variable selection method (VSM) to a near-optimal decision tree in the sense of the minimal number of nodes of the tree we present three variable selection criteria from different standpoints: **A** from combinatorial, **H** from entropy and **D** from discriminant analysis.

In e-cost case, the combinatorial criterion splits into three criteria **loss**, **Q** and **O**. Thus we have total 5 criteria together with e-cost versions of the criteria **D** and **H**.

Experimental results indicate that the combinatorial criteria show slightly better performance than others with the expense of auxiliary storage.

1. Introduction

A decision table (hereafter called a *table*) is a list of rules. A rule consists of a pair of conditions and an action, representing an "if-then-do" rule such that the action is to be executed when the conditions are satisfied. Such table is used very widely in the situation where an "input" object is required to be classified (identified) according to the values of its properties. This is a structure underlying in many areas including complexity theory [Bud85,Lov85,Weg84], identification of a specimen, diagnosis [Gar72], clustering, pattern recognition, programming and database searching [Han77]. One of the feasible ways to do this is to test each property sequentially one by one until an object is determined uniquely (*sequential test procedure STP*). This is usually represented by a decision tree (hereafter called *tree*). In most practical cases the table has many don't-care entries and the decision is possible by testing part of the properties. Thus automatic translation of a table into an optimum or a near-optimum tree under some optimality

criterion is desirable because there are many such trees (there can be at most $\prod_{i=1}^{L} i^{2^{i-1}}$ equivalent trees for an L-ary table f [ScS76]). The two optimality criteria adopted in this paper are: d-cost (i.e. number of the nodes of a tree) and an average cost of testing (an average weighted path length) over the tree.

Section 3 briefly describes the construction of optimum trees. Given an input table in completely expanded form (hence its size is 2^L, where L is the arity of the input table), the construction algorithm of optimum trees *always* requires $O(L3^L)$ operations with $O(3^L)$ storage [Bay73,ScS76]. The optimal variable theorem accelerates this algorithm in d-cost case but does not reduce its computation lower than $O(3^L)$.

On the other hand, a top-down method employing successive variable selections (VSM: variable selection method) can construct near-optimum trees in *at most* and usually far less than $O(L^2 2^L)$ operations. The VSM is applicable to a more practical case in which initial table is a partial function.

In Section 4 we propose three VSM criteria: **A** from combinatorial, **H** from entropy and **D** from discriminant analysis standpoints, for constructing a near optimum tree in the sense of the minimum number of the nodes of the tree (a special case of d-cost). We examine their formal properties: *nev*-free (rejection of nonessential variable) and *tev*- or *qdv*-bound (selection of a totally essential variable or a quasi-decisive variable; *tev*s are optimal under both costs and *qdv*s are also optimal under d-cost). We show that **A** is *nev*-free and *tev*-bound but not *qdv*-bound, while **H** and **D** have just the complementary property of this. The criteria **H** and **D** require at most $O(L^2 2^L)$ operations (bit comparisons), while the criterion **A** requires $O(L^2 2^L)$ operations (additions) with auxiliary storage of $O(L^2 2^L)$. Experimental results show that the performance of **D** and **H** practically coincides and the optimality attained by **A** is slightly better than those by **D** and **H** (1.03 vs. 1.05 compared with optimum trees).

In e-cost case, the combinatorial criterion **A** splits into the three variable selection criteria: **loss**, **Q** and **O**. Then we present e-cost versions of **H** and **D**. We show that **Q** and **O** are *nev*-free while the others are not. Experiments show that the "optimality coefficients" of **Q**-trees and **O**-trees are only 1.02 – 1.05 in some general case, while that of **loss**-trees is 1.10 – 1.14. The entropy criterion **H** is worse than **loss** and better than another simple heuristic **minc**. Moreover, the average numbers of nodes of the resulting trees of **Q** and **O** are only 1/4 of the corresponding **loss**-trees, indicating that the property of *nev*-free is crucial for a selection criterion. The performance of the three criteria **Q**, **O** and **loss** is at least not worse than the known heuristics at least on some example.

2. Definitions

Let $\{1,\ldots,L\}$ and $\{a_1,\ldots,a_K\}$ be the sets of L *properties* and K *actions*. Assume that each property i takes, for simplicity, the binary value $x_i = 0$ or 1, and the determination of the value incurs some cost. We are given a function $f : \{0,1\}^L \to \{a_1,\ldots,a_K\}$, called L-ary-K-action *decision table*, or simply a *table*, which maps the values of the properties into the actions. Let $I = \{\boldsymbol{x} | \boldsymbol{x} \in \{0,1\}^L\}$ be a set of *vectors* $\boldsymbol{x} = x_1 \ldots x_L$. A pair $(\boldsymbol{x}, f(\boldsymbol{x}))$ is called a *rule*. We treat a table f as the set of all 2^L rules. We assume that an a priori *distribution function* $p(\boldsymbol{x})$ of its execution probability is given over the set I.

2.1. Subtables and Fixations

Given an *initial table* f, a *subtable* is a restriction $f|(x_{i_m} = s_m \text{ for } m = 1,\ldots,h) = f(x_1 \ldots s_1 \ldots x_i \ldots s_h \ldots x_L)$ which is an $L-h$-ary function. The variables x_{i_m}, $m = 1,\ldots,h$, $0 \le h \le L$ are called *fixed* (to s_m), and the remaining variables are *free*. A subtable consists of all rules having vectors with some of their elements equal fixed constants. The number of free variables is the *arity* of the table (initial table is with the highest arity L, and a rule with the lowest arity 0). Since the initial table consists of 2^L rules, we have 2^{2^L} different subsets, among them 3^L subsets are subtables.

Hereafter property i and x_i are called *variable* and its *value* respectively. The sole type of restriction $f|(x_{i_m} = s_m)$ we deal with is called a *fixation of* f and its succession is denoted by $fi_1^{s_1} \ldots i_h^{s_h}$ or simply $f\alpha$, where α denotes a concatenation of $i_1^{s_1},\ldots,i_h^{s_h}$. The null fixation λ corresponds to the initial table f. We allow to treat fixation also as a "mask" operator to $\boldsymbol{x}$, i.e.

$$\alpha\boldsymbol{x} = x_1 \ldots x_{i_1-1} s_1 x_{i_1+1} \ldots x_{i_m-1} s_m x_{i_m+1} \ldots, x_L.$$

This enables us to write the rules of $f\alpha$ simply $\alpha I = \{\alpha\boldsymbol{x} | \boldsymbol{x} \in I\}$.

2.2. Decision Trees and Variable Selection Method (VSM)

Every tree we deal with is an *extended binary tree* [Knu73a, p.399]. Each node of a binary tree has exactly one *in-edge* (except root R which has no in-edge) and either zero or two *out-edges*. *External nodes* (leaves) are those with no out-edges. The remaining nodes of the tree are called *internal*. The *path* of a node is the set of nodes which are connected by the edges from the root R to the node.

A *decision tree* for f is a binary tree associated with each internal node a subtable and a variable (called *test variable*), and with each leaf a *decision action* according to the following algorithm:

If f is a constant (f consists of a single action) then the decision tree for f is a leaf with the decision action. Otherwise it is a tree having a root associated with the subtable f and with any its free variable i as the test variable, and having the left and right subtrees corresponding to the subtables fi^0 and fi^1, respectively (the edges leading to them are labeled by 0 and 1, respectively).

Node which contains a constant (nonconstant) subtable is called simply a *constant (nonconstant) node.* If we have some criterion to choose a test variable from the set of free variables, then we can construct a tree by repeating "select i from the variables of the subtable of a node according to the criterion and make its two sons which contain fi^0 and fi^1 respectively" for each nonconstant external node until no such node exists in the tree. This general algorithm of constructing a tree is called a *Variable Selection Method (VSM)* and its basic process consists of *dividing* a subtable. The number of non-constant subtables which appear in T is at most $2^L - 1$.

As we will see in Section 4 our VSM needs in the worst case $O(L^2 2^L)$ computation. Simple VSM by a random selection of a variable requires in the worst case $L2^L$ bit-test operations. This is easy to see because we have maximum 2^L rules at each level and there are maximum L levels (we need bit-test operations as many as the number of rules to divide a table).

One may think that each rule $\boldsymbol{x}$ of a table f is identified as belonging to fi^0 or fi^1 according to $x_i = 0$ or 1 respectively at each internal node, where i is a test variable of the node. A path of the tree represents such successive fixations. Thus a tree can be considered as a device to determine values of a given function by means of successive fixations. We call a decision tree simply a *tree* and denote it by T. A node of a tree is represented by $(r, i, f\alpha)$, where r is an identifier, i is a test variable and $f\alpha$ is a subtable of the node. This notation enables us to write a leaf by $(r, \lambda, f\alpha)$.

2.3. Description and Execution Costs of a Tree

Two different costs are associated with each variable i, $1 \leq i \leq L$: description cost (abbreviated by *d-cost*) C_i^d and execution cost (abbreviated by *e-cost*) C_i^e, through which we define two costs for a tree.

We define *d-cost of a tree* T by

$$|T|^d = \sum_{(r,i,*) \in T} C_i^d, \tag{1}$$

where the summation is taken over all internal nodes of the tree. Thus $|T|^d$ is the cost for representing the whole test procedures of T and does not depend on its execution.

"Average" cost of an execution of a tree depends on the execution frequencies of the rules $\{(\boldsymbol{x}, f(\boldsymbol{x}))\}$, which we assume a prior known. The distribution function $p(\boldsymbol{x})$ is the normalized execution frequencies of rules, i.e. $p(I) = \sum_{\boldsymbol{x}\in I} p(\boldsymbol{x}) = 1$ and $p(\boldsymbol{x}) > 0$ for each $\boldsymbol{x}$. Then for any subtable $f\alpha$, its execution probability, i.e. the probability that one of the rules in the subtable is executed, is computed by $p(f\alpha) = p(\alpha I) = \sum_{\alpha\boldsymbol{x}\in\alpha I} p(\alpha\boldsymbol{x})$. Obviously $p(f) = 1$.

Along a path to a node $(r, i, f\alpha)$, we test h fixed variables of $f\alpha$ for a total cost of $fixcost(f\alpha) = \sum_{m=1}^{h} C^e_{i_m}$, which we call *fixation cost* of $f\alpha$ (this corresponds to the *weighted path length* of the node).

The *e-cost* of a tree is defined by the expectation of the fixation cost for all leaves of the tree, i.e.

$$|T|^e = \sum_{(r,\lambda,f\alpha)\in T} p(f\alpha) fixcost(f\alpha), \tag{2}$$

where the summation is taken over all the leaves (external nodes) of T. Thus $|T|^e$ represents an average cost of testing required for deciding an action through the tree. An alternative expansion form of the e-cost is $|T|^e = \sum_{(r,i,f\alpha)\in T} p(f\alpha) C^e_i$, where the summation is taken over all internal nodes of T.

2.4. Optimum Decision Trees

We call a tree *optimum* when its cost is a minimum among all trees corresponding to the table f.

Example 2.1.

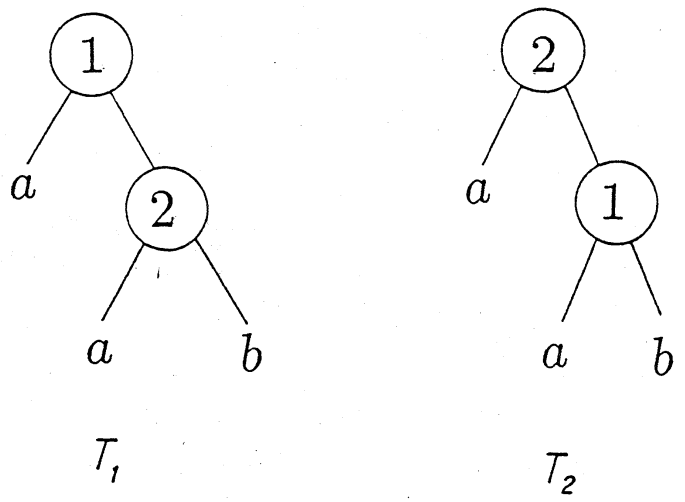

Fig. 2.1.

x_1x_2	action	probability
00	a	p_0
01	a	p_1
10	a	p_2
11	b	$p_3 = 1 - (p_0 + p_1 + p_2)$

	$\|T_1\|$	$\|T_2\|$
d-cost	$C_1^d + C_2^d$	$C_2^d + C_1^d$
#nodes	2	2
e-cost	$C_1^e + (p_2 + p_3)C_2^e$	$C_2^e + (p_1 + p_3)C_1^e$

The possible trees are shown in Fig. 2.1. Their respective costs are given in the table. Note that $|T_1| \leq |T_2| \Leftrightarrow$ variable 1 is optimal.

3. Construction of an Optimum Decision Tree

A construction algorithm of an optimum tree is known [Bay73] which *always* requires $O(L3^L)$ comparisons where L is the arity of the input table. The quantity $O(3^L)$ comes from the number of all subtables.

The algorithm is based on the dynamic programming principle. It computes an optimum cost (tree) for every subtable by comparing the costs of "quasi-optimum" trees having the test variable i, where i runs all the free variable of the subtable.

We measure the amount of computation needed for the algorithm by the number of *min*'s (comparisons of two costs). We have

$$3^L - 2^L \leq \text{computation} \leq (L/3)3^L.$$

The lower bound is attained when *min* is performed once for each subtable except arity 0 subtables (we don't know an algorithm which attain this lower bound) and the upper bound when *min* is computed for all free variable i of the subtable. Thus the upper bound is the amount of computation always required by the dynamic programming algorithm. Since the size of the initial input table is about $N = 2^L$, the number of the operations needed for this algorithm is $O(N^{log3}logN) = O(N^{1.6}logN)$, where the *log* has base 2. The optimal variable theorem (Theorem 4.3) can accelerate this algorithm in d-cost case (and not in e-cost case) because it enables us to determine optimal variable without running i for all free variables in some of the subtables. The amount of this reduced computation lies somewhere between the two bounds. As we have seen, the computational order cannot be lowered below $O(N^{log3}) = O(N^{1.6})$ by the acceleration. However, it greatly reduces the computation. For example, for 10 variable 4 action tables, where the occurrences of each action obey 80-20 rule [Knu73b, p.397], the computation decreases to 42 % its value without acceleration, while the total execution time decreases to about 80 %.

The dynamic programming approach for constructing optimum decision trees are reported by many authors [ScS76,MiS80]. Another construction algorithm of an optimum tree by branch-and-bound principle is reported in [ReS66,ReS67].

4. Three VSM Criteria in D-Cost Case

Average computation of VSM is expected to be much less than its worst case evaluation $O(L^2 2^L)$ ("complete tree" is the worst case and most decision trees in practice are far from being complete). We are going to propose such efficient VSM criteria. At the same time, we want to claim something formally about the performance of our criteria. For example, one wants to know which is the best criterion among them or in what situation each criterion is adequate. For this purpose we first investigate conditions of "optimum" or "worst" selection, then examine whether our criteria satisfy these conditions or not.

4.1. Optimal Variables and Nonessential Variables

We need a notation to represent a unary subtable also called an *i-pair* of f. This is a subtable whose all variables except i are fixed. Let us denote a subtable simply by f.

1. Let f have all h variables denoted by $1,\ldots,h-1$ and i $(i \neq k, k = 1,\cdots,h-1)$. Let $u = u(1)\cdots u(h-1)$, $u(k) = k$, $k = 1,\ldots,h-1$, denote a sequence (i.e. concatenation) of $1,\ldots,h-1$.

2. Let $x = x(1)\ldots x(h-1)$, $x(k) = 0$ or 1 for $k = 1,\ldots,h-1$, denote a bit sequence of length $h-1$.

Denote a fixation $u(1)^{x(1)}\ldots u(h-1)^{x(h-1)}$ by u^x for simplicity. Let $u^x i$ represent a vector in which the variables in u are fixed to the values x and only the variable i is free.

Then an i-pair is represented by $f(u^x i) \equiv f u^x(u^x i)$, which consists of a pair of rules of f: $(u^x i^0, f(u^x i^0))$ and $(u^x i^1, f(u^x i^1))$. If two actions of an i-pair coincide, it is a *constant (inactive) i-pair*, otherwise it is a *nonconstant (active) i-pair*. Each fixation u^x which gives inactive or active i-pair is called *inactive* or *active* fixation for i, respectively (hereafter only x is indicated instead of u^x since u is the "complement" sequence of i).

Example 4.1. For $x = 00$ and $u = 23$ the fixation u^x denotes $2^0 3^0$. Then 1-pair $f(2^0 3^0 1) = f 2^0 3^0(2^0 3^0 1)$ for f in Table 4.1 is a nonconstant 1-pair, while $f(1^1 2^0 3) = f 1^1 2^0(1^1 2^0 3)$ is a constant 3-pair. Hence the fixation $2^0 3^0$ is active for 1, while $1^1 2^0$ is inactive for 3 in f.

Now we define some notions about variables. Recall that the probability of each rule is not 0. A variable i is *nonessential* if each fixation x for i is inactive, i.e. $f(u^x i) = a_j$ for each x (a constant action a_j is determined depending on x). Equivalently i is *essential* if there is an active i-pair. Again, a variable i is *totally essential* if each fixation x for i is active, i.e. $f(u^x i) \neq const.$ for each x. Finally, let us call i *quasi-decisive* if $fi^s = const.$ and $fi^{\bar{s}} \neq const.$ for strictly either one of $s = 0$ or 1 [Miy85]. Nonessential, totally essential and quasi-decisive variables are abbreviated to *nev*, *tev* and *qdv*, respectively. Note that when rules with probabilities 0 are allowed, these definitions should be modified in an appropriate way.

Let us call a variable i *optimal* if there is an optimal tree having i as a test variable at the root. This means that we have an optimal tree having optimal left and right subtrees for fi^0 and fi^1, respectively.

Theorem 4.1. *Non-essential variables cannot be* optimal *under both cost. That is, a nonessential variable never appears as a test variable in an optimum tree.*

Theorem 4.2. (cf.[GaR73]) *Total essential variables are* optimal *under both costs.*

Theorem 4.3. [Miy85] Optimal Variable Theorem. *Qdvs are optimal under d-cost. Conversely, if there are qdvs, then only they are optimal under d-cost.*

Note that a *qdv* is not an optimal variable in general with respect to e-cost.

Therefore it is desirable for a good criterion to reject *nevs* and select a *tev* or *qdv*. Let us call a criterion *nev-free* if it doesn't select an *nevs*. Again, let us call a criterion *tev-bound* or *qdv-bound* if it selects a *tev* or *qdv* whenever they exist. Now we present three criteria **A**, **H** and **D** for selecting a variable to construct an efficient trees in the sense of minimal number of the nodes of the tree and see that they satisfy these properties in a complementary way.

Our basic strategy is "to make constant tables as fast as possible" since no more dividing is necessary for constant tables. Hence our criteria should reflect both distances between fi^0 and a constant function and fi^1 and a constant function.

4.2. Activity Criterion A

Define $A_i :=$ the number of active i-pairs of f. An active i-pair is a logical unit to be separated by dividing a table. Then $A = \sum_{i=1}^{L} A_i$ represents the total number of active pairs of rules to be separated. Since A_i is lost by dividing f by i, we select a variable i which have a maximum number of A_i among all variables (cf. [Spr66,MiO82]). This *activity criterion* we distinguish by **A**.

Proposition 4.1. *Activity ranges* $0 \leq A_i \leq 2^{L-1}$. *The* best *value* $A_i = 2^{L-1}$ *is attained if and only if* i *is a tev, and the worst value* $A_i = 0$ *is attained if and only if* i *is a nev.*

4.3. Entropy Criterion H

The VSM procedure can be "viewed" as a process of perpetual increase of the determinacy (equivalently, decrease of the ambiguity) of actions until finally we get all tables completely determined [HVMG82,MiO82,Mor82]. To describe determinacy we first give some notations.

We denote by N_j the number of the occurrence of the action a_j in f, and let

N_j^{i0} := the number of rules $(\boldsymbol{x}, f(\boldsymbol{x}))$ such that $f(\boldsymbol{x}) = a_j$ and $x_i = 0$, and similarly
N_j^{i1} := the number of rules $(\boldsymbol{x}, f(\boldsymbol{x}))$ such that $f(\boldsymbol{x}) = a_j$ and $x_i = 1$.

We have the following equations:

$$\begin{aligned}
&1)\ N_j^{i0} + N_j^{i1} = N_j,\\
&2)\ \textstyle\sum_j N_j^{is} = N^{is} = 2^{L-1} = N/2,\ s = 0, 1,\\
&3)\ \textstyle\sum_j N_j = N = 2^L,\\
&4)\ p_j = N_j/N\ (\textstyle\sum_j p_j = 1),\\
&5)\ p_j^{is} = N_j^{is}/N^{is} = 2N_j^{is}/N.
\end{aligned}$$

The actions a_j in a table f can be considered to occur with the probabilities p_j, $j = 1, \ldots, K$. Therefore the nondeterminacy (ambiguity) can be measured by the entropy defined by

$$H(f) := -\sum_{j=1}^{K} p_j \log\ p_j. \tag{3}$$

This gives the following *entropy criterion* [MiO82]: select a variable i which has the least value of

$$H_i := -1/2 \sum_{s=0,1} \sum_{j=1}^{K} p_j^{is} \log\ p_j^{is}, \tag{4}$$

where p_j^{is} is the probability of a_j in fi^s $(s = 0, 1)$.

Proposition 4.2. *The entropy* H_i *ranges* $0 \leq H_i \leq \log K$. *The* best *value* $H_i = 0$ *is attained if and only if either* f *is a constant or* i *is a unique essential variable of* f, *and the* worst *value* $H_i = \log K$ *is attained if and only if each action* a_j *occurs equiprobably for* $j = 1, \ldots, K$ *in both subtables* fi^s *for* $s = 0$ *and 1.*

4.4. Discriminant Criterion D

Each variable x_i ($x_i = 0$ or 1) contributes to discriminate different actions and this can be measured by "discriminating power" of a variable x_i from the standpoint of

the discriminant analysis [Fis36], which presents the following ratio of variances as its measure [Wil62]:

$$\eta_i = \sigma^2_{Bi}/\sigma^2_{Wi}, \tag{5}$$

where σ^2_{Bi} and σ^2_{Wi} represent the between-action (interclass) and the within-action (intraclass) variances of the variable x_i, respectively. They are given by

$$\sigma^2_{Bi} = \sum_j w_j(\mu^i_j - \mu^i_T)^2, \tag{6}$$
$$\sigma^2_{Wi} = \sum_j w_j(1/N_j) \sum_{x \in class(j)} (x_i - \mu^i_j),$$

where μ^i_j and μ^i_T denote the mean values of x_i with respect to the action a_j and f, respectively. We have:

6) $\mu^i_j = E_{a_j}\ x_i = \sum_{a_j} x_i/N_j = N^{i^1}_j/N_j$,
7) $\mu^i_T = E\ x_i = \sum x_i/N = \sum_j N^{i^1}_j/N = N/2 \cdot 1/N = 1/2$.

One may wonder that the values 0 and 1 assumed by a variable x_i are "nominal" entities only to be used to distinguish two different things. However, we can use 0 and 1 in place of any two different real numbers. This is because the η_i so defined by a ratio of two variances is invariant under an affine transformation of coordinate x to $y = b(x+a)$, i.e. shift a and scale factor b; in other words from $x = 0, 1$ to $y = ab,\ b(1+a)$.

Since $\sigma^2_{Bi} + \sigma^2_{Wi} = \sigma^2_{Ti}$ and $\sigma^2_{Ti} = 1/4$ (from $\mu^i_T = E\ x_i = 1/2$), the interclass variance $\sigma^2_{B_i}$ alone represents the degree of separation of the classes, giving a criterion [MiO82]: select a variable i which has the greatest value of

$$\sigma^2_{Bi} = (1/N)\sum_j (N^{i^1}_j)^2/N_j - 1/4. \tag{7}$$

Proposition 4.3. *The σ^2_{Bi} ranges $0 \le \sigma^2_{Bi} \le 1/4$. The* best *value $\sigma^2_{Bi} = 1/4$ is attained if and only if i divides the action set into two disjoint sets, i.e. the actions of fi^0 and fi^1 have no action in common, and the* worst *value $\sigma^2_{Bi} = 0$ is attained if and only if i divides each action class into two halves.*

In **Table 4.1** we give the values of the above variable selection criteria for the function given there. All the criteria select 1 or 3 as a first test variable and can give a minimum tree.

Table 4.1.

$f(123)$

123	action
000	b
001	c
010	a
011	c
100	a
101	a
110	a
111	b

variable	1	2	3
A_i	3	2	3
μ_a^i	3/4	2/4	1/4
μ_b^i	1/2	1/2	1/2
μ_c^i	0	1/2	1
σ_{Bi}^2	3/32	0	3/32
$p_a^{i^0}$	1/4	2/4	3/4
$p_b^{i^0}$	1/4	1/4	1/4
$p_c^{i^0}$	2/4	1/4	0
$p_a^{i^1}$	3/4	2/4	1/4
$p_b^{i^1}$	1/4	1/4	1/4
$p_c^{i^1}$	0	1/4	2/4
H_i	1.16	1.50	1.16

Theorem 4.4. **A** *is* nev-free *and* tev-bound *but not* qdv-bound, *while* **H** *and* **D** *have just the complementary properties.*

The results are summarized in the following table.

criterion		*nev*-free	*tev*-bound	*qdv*-bound
A	max A_i	○	○	×
H	min H_i	×	×	○
D	max σ_{Bi}^2	×	×	○
optimum		○	○	○

4.5. Amount of computation of the criteria

Dividing a table f by a variable i is done by testing i-th bit of each rule and then transferring the rule into the corresponding subtree (either fi^0 or fi^1) according to its value 0 or 1, respectively. Thus we need bit-test operations as many as the number of rules of a table for dividing a table (we also need the same number of transfer of a rule). Thus a VSM need at least $O(L2^L)$ amount of computation for simply constructing a tree whatever criterion is used for the variable selection (in the worst case). We additionally need computation for choosing a test variable, which usually requires amount of computation larger than proportional to the number of the rules. So this part is dominant in the computation.

The most time-consuming part is to calculate $N_j^{i^0}$ and $N_j^{i^1}$. For this we need two operations: another bit-test and an operation of table look-up which returns an action class j for a given $\boldsymbol{x}$ such that $f(\boldsymbol{x}) = a_j$. This is done for each of $L - l$ free variables at each level l. Thus we need $2^L \sum_{l=0}^{L-1}(L-l) = L(L+1)2^{L-1}$ bit-test operations for the

whole tree. We need table look-up operations exactly a half amount of that for bit-test operation, since $N_j^{i0} = N_j - N_j^{i1}$ and N_j may be assumed to be known beforehand. However, note that in a sense **D** is more advantageous than **H** because it is represented only by N_j^{i1} (and N_j).

For the computation of activity of i we take another approach. We initially generate all active i-pairs of an initial table f for all $i = 1, \ldots, L$ and store them in an array ALIST$(i), i = 1, \ldots, L$. Then each subtable inherit pointers to ALIST(i) for its active i-pairs from its "mother" table. The computation for this pointer manipulation is evaluated in the worst case to the number of $L(L-1)2^{L-2}$. The storage necessary for the pointers to ALIST is given in the worst case by $L(L+1)2^{L-2}$ [MiO87].

The above worst-case evaluations of the amount of computation for the criteria are summarized in the following table. In our implementation **A** requires $O(L^2 2^L)$ auxiliary storage to save active pairs. We also give average time for constructing a tree, where $L = 10$, $K = 4$ and average is taken over 1000 tables. This experiment was performed on a FACOM M380 computer executing 12 MIPS, with 8M bytes of available main memory.

criterion	computation	average time (second)
random	$O(L2^L)$	1.02
H	$O(L^2 2^L)$	2.01
D	$O(L^2 2^L)$	2.03
A	$O(L^2 2^L)$	2.59
	with additional storage $O(L^2 2^L)$	
optimum (DP)	$O(L3^L)$	16.60

4.6. Experimental Observations

We investigated the performance experimentally on random tables for $K = 2$–32 and $L = 4, 6, 8, 10$, where the occurrence of actions is artificially weighted. Experimental results show that the performance of H and D practically coincides and is slightly worse than **A** (more precisely, 1.05 vs. 1.03 in terms of "optimality coefficient", i.e. average of optimum trees is taken as 1.0).

5. Five VSM Criteria in E-Cost Case

In e-cost case, the criterion **A** splits into the three criteria: **Q, loss** and **O**.

5.1. A Criterion Loss

The relative probability of the nonconstant i-pairs contained in f, i.e.

$$a_f^+(i) := (1/p(f) \sum_{active\ x} p(f(u^x i)) \tag{8}$$

is called *activity* of i with respect to f, where $p(f)$ denotes the sum of the probabilities of the rules in f.

A nonconstant i-pair can be separated only by the test variable i. Hence, $a_f^+(i)$ represents a degree of "essentiality" of dividing f by i. On the other hand, $a_f^-(i) := 1 - a_f^+(i)$ represents a ratio of separating constant i-pairs by dividing by i, and hence this may include some "potentially inessential" portion. Thus

$$loss_f(i) := C_i(1 - a_f^+(i)) \tag{9}$$

is called *loss of testing the variable i with respect to f.* A near-optimum tree might be constructed if we choose i which has a minimum value of $loss_f(i)$ among all variables of f [MTG80]. We distinguish this criterion by **loss**.

We note that another simple VSM is to select a variable i which has a minimum cost C_i among all free variables. We distinguish this criterion by **minc**.

5.2. A Criterion Q Based on a Potential

In [Miy87] we have introduced the notion of a "potential" of a table and have formulated a simple but intuitive scheme for deriving a criteria from it. A potential represents a certain measure of "complexity" of a table. It is shown that, for a subtable f having all free variables $1, 2, \ldots, h$,

$$q(f) := \sum_{i=1}^{h} a_f^+(i). \tag{10}$$

is a potential (called *activity potential*). Then the following *activity potential criterion* is derived from the standpoint of maximizing the decrease of the potential of the tree caused by the splitting of the node (per unit cost). A VSM **Q** [Miy87] selects a variable i which has a maximum value of

$$Q_f(i) := a_f^+(i)/C_i \tag{11}$$

among all variables of f.

5.3. A Combined Criterion O

Combining the two criteria **loss** and **Q**, let the third criterion **O** [Miy87] be: select a variable i that has a maximum value of

$$O_f(i) := a_f^+(i)/C_i^2(1 - a_f^+(i)). \tag{12}$$

We immediately see that if $C_i = const.$ for all i, then all the three criteria select the same variable, since each criterion is a monotone function of $a_f^+(i)$. It is known that the criterion **loss** always selects an optimal variable when (and only when) $L = 2$ [Mor82], while **Q** and **O** do not always select an optimal variable even when $L = 2$.

5.4. Entropy Criterion

In e-cost case the entropy criterion **H** reduces to selecting i which gives a minimum of

$$H_i := \sum_{s=0,1} (-\sum_{j=1}^{K} r_j^{i^s} \log r_j^{i^s} + p(fi^s) \log p(fi^s)), \tag{13}$$

where $r_j^{i^s}$ denotes the sum of the probabilities of the rules of the class j in the subtable fi^s.

5.5. Discriminant Analysis Criterion

The discriminant analysis criterion **D** reduces to selecting i which gives a maximum of

$$\eta_i := (\sum_j (r_j^{i^1})^2/r_j - p(fi^1)^2/p(f))/(p(fi^1) - p(fi^1)^2/p(f)), \tag{14}$$

where r_j denotes the sum of the probabilities of the rules of the class j in the table f.

Note that we have no $\sigma_{Ti}^2 = 1/4$ for all i, so we need the full computation of η_i in contrast to the d-cost case (where the computation of σ_{Bi}^2 is sufficient).

5.6. Properties of the Criteria

Similar analysis as the d-cost case has been done over these five criteria. Their properties about *nev*-free and *tev*-bound are given in the following table. It is rather hard to conclude that some criterion is superior to the others from their formal properties (note that **O** doesn't have a property called "monotonicity" which **Q**, **loss** and optimum criterion have) [Miy87].

criterion		*nev*-free	*tev*-bound
minc	min C_i	×	×
loss	min $loss_i$	×	○
Q	max Q_i	○	×
O	max O_i	○	○
H	min H_i	×	×
D	max η_i	×	×
optimum		○	○

5.7. Amount of Computation and Implementation

To compute $\sum_{active\ x} p(f(u^x j))$ we need sum operations at most proportional to the number of rules of the subtable f (the value $p(f)$ is computed once when it is generated and kept). This is done for each free variable of f. Hence the amount of computations and storage required for the algorithm are $O(L2^L)$ and $O(L^2 2^L)$. Similarly as we have seen before we need $O(L^2 2^L)$ extra storage for the list of non-constant pairs of rules.

5.8. Experimental Observations and Performance

The performance of the three criteria **Q, O** and **loss** are tested on two classes of artificial data q-d tables and 8-2 tables for $L = 4 - 12$, and compared to optimum trees (constructed by DP for $L = 4 - 10$) and **minc** trees in [Miy87]. A q-d tables consists of all q-d variables (its tree is a chain) and an 8-2 table consists of two actions occurring with 8:2 ratio. Further both the costs of the variables and execution probabilities are weighted according to the well-known 80-20 rules [Knu73a, p.397].

In q-d table **loss** shows the best result. This is because *nev* selection does not occur in q-d tables. On the contrary, for a more general 8-2 tables, **loss** showed $7-10$ % worse performance compared to **Q** and **O** because of *nev*-selections (% is with respect to the average cost of optimal trees). Of more significance is that its *nev*-selection invokes 4 times bigger trees compared to **Q** and **O** (constructing the tree required twice as much time). This shows that *nev*-free property is extremely important for a selection criterion. The performance of **Q** and **O** are 2 – 3 % worse than optimal and **O** is slightly better than **Q** in all the cases. The performance of entropy criterion is worse than **loss** and better than **minc**.

As for the worst case of the criteria, a case constructed on uniform variable cost [MTG80] shows an asymptotically $L - 1$ optimality coefficient when an appropriate probability distribution is assumed for the uniform cost case. Any of **loss**, **O** and **Q** are not worse than other known heuristics on the real practical table from the business data processing area given in [Ver72].

6. Discussions and Conclusions

Tables (functions) and trees (its implementations) appear in many applications. They are basic tools for controlling information. Tree also provides an alternative way to compute values of a function. Then optimal tree represents an optimal computation of a function. In this paper we have surveyed the construction algorithms

of optimum or near-optimum trees along the lines presented or to be presented in [Miy85,Miy87,MiO87]. The problem treated here may be too simple to be applied to many applications in "knowledge engineering" where "if-then-do" rules play an important role. In practical case, the given initial table is a partial function, i.e. the rules are not exhaustive. An extension of the criteria to this case is given in [Miy87]. In reality, actions are given usually as a set of vectors of actions, and there given a set of initial tables instead of a single table. Some applications of the optimization can be seen in [Bar79,Cha86,Sas85].

The optimization problems are NP-complete when input tables are not expanded form [HyR76,MiS80,MTG81]. The input size 2^L for a completely expanded table makes the DP algorithm superficially polynomial order of computation. Hence various algorithms constructing near-optimum trees and their performance studies are of significance. Especially, a good framework (not ad hoc one) is most desirable for treating tables represented in reduced form (using "don't-cares"). Although their performance is inferior to combinatorial ones, statistical approach (discriminant analysis) deserves to be studied more carefully, because it involves less computation. Some bottom-up heuristics for constructing near-optimum trees could be devised. It is an open problem whether $O(3^L)$ is the best possible for constructing an optimum tree. It is also an open problem to find another "optimal variable theorem" or its refinement.

Acknowledgements

The author is truly indebted to Dr. Nobuyuki Otsu whom he owes the idea of discriminant analysis and many discussions. He thanks to Drs. Koichiro Tamura, Jiro Ihara, Shinji Umeyama, Takio Kurita and Hideki Asoh for many helpful and stimulating discussions.

References

[Bar79] Baranov, S.I.: Synthesis of microprogrmmable automata (Russian), Leningrad, Energia, 1979, 1-232.

[Bay73] Bayes A.J.: A dynamic programming algorithm to optimise decision table code. Australian Computer J. 5, 2 (May 1973), 77-79.

[Bud85] Budach L.: A lowere bound for the number of nodes in a decision trees, Electron Inf. verarb. Kybern. EIK **21** (1985) 4/5, 221-228.

[Cha86] Chan, A. H.: Using decision trees to derive the complement of a binary function with multiple-valued inputs, IEEE Trans. on Comput., C-36, 2, 1987, 212-214.

[Fis36] Fisher, R.A. The use of multiple measurements in taxonomic problems, Ann. Eugenics, 7, Part II, 179-188 (1936).

[Gar72] Garey, M.R.: Simple Binary identification problems. IEEE Trans. Comput. TC-21, 6, 588-590, June 1972.

[GaR73] Ganapathy, S., Rajaraman, V.: Information theory applied to the conversion of decision tables to computer programs. Comm. ACM, 16, 9, 532-539, Sept., 1973.

[HVMG82] Hartman, C.R.P., Varshney, P.K., Mehrotra,. K.G., Gerberich, C.L.: Application of information theory to the construction of efficient decision trees. IEEE Trans. on Infomation theory it-28,4, 565-577, July 1982.

[HyR76] Hyafil, L., Rivest, R.L.: Constructing optimal binary decision trees is NP-complete. Comm. ACM 5, 1, 15-17, May 1976.

[Knu73a] Knuth, D.E.: The art of computer programming, vol. 1, 2nd ed. Reading, Mass.: Addison-Wesley, 1973.

[Knu73b] Knuth, D.E.: The art of computer programming, vol. 3, Reading, Mass.: Addison-Wesley, 1973.

[Lov85] Loveland, D. W.: Bounds for binary testing with arbitrary weights, Acta Informatica 22, 101-114, 1985.

[MiS80] Miyakawa, M., Sabelfeld, V.K.: On minimizations of size of logical schemes (in Russian). Theoretical basis of compiling (A. P. Ershov, ed.), 49-58, Novosibirsk State University, Novosibirsk 1980.

[MiO82] Miyakawa, M., Otsu, N.: Algorithms constructing near-minimum total nodes decision trees from expanded decision tables (in Japanese). TGEC IECE Japan, EC82-33 ,July 1982.

[Miy85] Miyakawa, M.: Optimum decision trees - an optimal variable theorem and its related applications -. Acta Informatica 22, 475-498, 1985.

[Miy87] Miyakawa, M.: Criteria for selecting a variable in the construction of efficient decision trees, IEEE Trans. Comput., to appear.

[MiO87] Miyakawa, M., Otsu, N.: Three criteria for selecting variables in near-optimum decision tree construction, manuscript 1987.

[MTG80] Moret, B.M.E., Thomason, M.G., Gonzalez, R.C.: The activity of a variable and its relation to decision table. ACM Trans. Prog. Lang. Syst. 2,4, 580-595, Oct. 1980.

[MTG81] Moret, B.M.E., Thomason, M.G., Gonzalez, R.C.: Optimization criteria for decision trees, Dept. of Computter Science, Collage of Engineering,Univeersity of New Mexico, Technical Report CS81-6, 1981.

[Mor82] Moret, B.M.E.: Decision trees and diagrams. Computing Surveys 14, 4, 593-623, Dec. 1982.

[ReS66] Reinwald, L.T., Soland, R.M.: Conversion of limited-entry decision tables to optimal computer programs I: minimum average processing time. JACM 13, 3, 339-358, July 1966.

[ReS67] Reinwald, L.T., Soland, R.M.: Conversion of limited-entry decision tables to optimal computer programs II: minimum storage requirement. JACM 14, 4, 742-755, Oct. 1967.

[Sas85] Sasao, T.: An algorithm to derive the complement of a binary function with multiple-valued inputs, IEEE Trans. on Comput., c-34, 2, Fubruary 1985, 131-140.

[ScS76] Schumacher, H., Sevcik, K.:The synthetic approach to decision table conversion. Comm. ACM 19, 6, 343-351, June 1976.

[Spr66] Sprague, V. G.: On storage space of decision trees. Comm. ACM 9, 5, 319-319, May 1966.

[Ver72] Verhelst, M.: The conversion of limited-entry decision tables to optimal and near optimal flowcharts: two new algorithms. Comm. ACM 15, 11, 974-980, November 1972.

[Weg84] Wegner, I.: Optimal decision trees and one-time-only branching programs for symmetric Boolean functions. Information and Control **62**, 129-143 (1984).

[Wil62] Wilks, S. S., Mathematical Statistics, John Wiley and Sons, New York, 1962, p.573.